Crowdsourcing Programming Assignments with CrowdSorcerer

Pirttinen, Nea

ACM
2018-07-02

Pirttinen , N , Kangas , V , Nikkarinen , I , Nygren , H , Leinonen , J & Hellas , A 2018 , Crowdsourcing Programming Assignments with CrowdSorcerer . in Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education . ACM , New York, NY , pp. 326-331 , 23rd Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018) , Larnaca , Cyprus , 02/07/2018 . https://doi.org/10.1145/3197091.319

http://hdl.handle.net/10138/315988
https://doi.org/10.1145/3197091.3197117

acceptedVersion

Crowdsourcing Programming Assignments with CrowdSorcerer

Nea Pirttinen
University of Helsinki
Helsinki, Finland
nea.pirttinen@cs.helsinki.fi

Vilma Kangas
University of Helsinki
Helsinki, Finland
vilma.l.kangas@helsinki.fi

Irene Nikkarinen
University of Helsinki
Helsinki, Finland
irene.nikkarinen@helsinki.fi

Henrik Nygren
University of Helsinki
Helsinki, Finland
henrik.nygren@helsinki.fi

Juho Leinonen
University of Helsinki
Helsinki, Finland
juho.leinonen@helsinki.fi

Arto Hellas
University of Helsinki
Helsinki, Finland
arto.hellas@cs.helsinki.fi

ABSTRACT

Small automatically assessed programming assignments are an often used resource for learning programming. Creating sufficiently large amounts of such assignments is, however, time consuming. As a consequence, offering large quantities of practice assignments to students is not always possible. CrowdSorcerer is an embeddable open-source system that students and teachers alike can use for creating and evaluating small automatically assessed programming assignments. While creating programming assignments, the students also write simple input-output -tests, and are gently introduced to the basics of testing. Students can also evaluate the assignments of others and provide feedback on them, which exposes them to code written by others early in their education. In this article we both describe the CrowdSorcerer system and our experiences in using the system in a large undergraduate programming course. Moreover, we discuss the motivation for crowdsourcing course assignments and present some usage statistics.

CCS CONCEPTS

• **Information systems** → *Crowdsourcing*; • **Human-centered computing** → *Collaborative content creation*; • **Social and professional topics** → *Computing education*;

KEYWORDS

crowdsourcing, automated assessment, assignment creation, peer review, programming

ACM Reference Format:
Nea Pirttinen, Vilma Kangas, Irene Nikkarinen, Henrik Nygren, Juho Leinonen, and Arto Hellas. 2018. Crowdsourcing Programming Assignments with CrowdSorcerer. In *Proceedings of 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3197091.3197117

ITiCSE'18, July 2–4, 2018, Larnaca, Cyprus

ACM ISBN 978-1-4503-5707-4/18/07…$15.00
https://doi.org/10.1145/3197091.3197117

1 INTRODUCTION

One of the recent trends in education has been the transition towards shared educational resources and ownership of education; instructors may collaborate to create shared educational resources [19, 20] and students can create support content to courses [4]. At an extreme, the students may even drive the course, deciding the topics and directions that they wish to pursue [7].

In this work, we study the possibility of using novice programmers as a crowdsourcing resource for generating programming assignments in the very first weeks of their first programming course. Our belief is that when students are given the opportunity and support to create programming assignments, they reflect on prior tasks and content, consider how they would themselves solve the problem they are creating, and also learn to articulate what it means for a program to be correct (or incorrect). Moreover, they may put themselves into the role of an instructor as they contribute towards the course, which can increase retention through a feeling of not only being a consumer but also a producer in the learning community [1]. Finally, as they also read and comment the content generated by other participants, they are exposed to code written by others early, which has been proposed to be helpful for novices' understanding of source code [22].

For the purposes of this work, we have generated an embeddable component for online course materials that makes it straightforward to generate assignments for programming courses. The system, called CrowdSorcerer, provides students a view for introducing the problem statement, model solution, and a set of simple test cases that are validated using a server. Students are also given the opportunity to review and comment on the assignments created by others.

The closest work to our research is PeerWise [4], which provides students the opportunity of generating multiple choice questions that can then be shared to other course participants, and CodingBat [17], CloudCoder [16], and CodeWrite [5], which are online systems for programming practice that provide some support for exercise authoring. Our work, however, drills down to the explicit generation and evaluation of programming assignments, which then could be used as a part of various platforms that are used for teaching and learning programming.

tThis article is organized as follows. In the next section, we briefly go over previous work on crowdsourcing in the educational context. Then, we outline the architecture and design of CrowdSorcerer in Section 3 and explain how it is used to both create and

review programming exercises. Then, in Section 4 we present our experiences of using the tool for the first time. Lastly, we discuss the tool in Section 5 and conclude the article in Section 6.

2 CROWDSOURCING IN EDUCATION

One of the first mentions of the term crowdsourcing appeared in the Wired magazine in 2006, where Howe discussed tapping to the latent talent of the crowd [10]. Since then, the term has become synonymous with almost any collaborative activity, where typically a large user base provides ideas and services to an organization or some other party [8]. In the context of the world-wide web, the term has been used for anything ranging from evaluation tasks (e.g. reviews, voting) to artifact or idea sharing, as well as for social networking and creating artifacts such as knowledge bases and software [6].

Perhaps the best example of crowdsourcing is Wikipedia [25], which is a free online encyclopedia that anyone with a free account can edit. In early 2018, there were over 30 million Wikipedia accounts and over 5.5 million articles in English, and millions more in other languages. Each account in Wikipedia can be used to edit and create articles, while reading the articles is free for anyone and requires no registration.

Crowdsourcing has been used to generate various types of educational content ranging from online books [20] and book supplements [9] to course-specific rehearsal content such as multiple choice questions [4]. Course participants in various institutions and MOOCs evaluate assignment responses that the participants themselves have submitted as a part of their coursework – research suggests that peers can give actionable and accurate feedback [13].

Crowdsourcing in education is not limited to artifacts generated in courses, but can also be seen in community efforts and tools. For example, one could view the ITiCSE Working Group "Canterbury QuestionBank" [19] from 2013 as an example of a small-scale crowdsourcing effort, where researchers and teachers have generated practice content for programming courses. Similarly, using peer assessment and tools that support peer assessment, which are reviewed extensively in [15], can be seen as crowdsourcing.

In the same vein, crowdsourcing is not limited to the artifacts that are created actively. The majority of data-driven approaches in educational data mining and learning analytics rely on data that is generated by students either actively or passively as they take part in a course. Such data has been used, for example, to improve learning materials [14], create predictive models that can be used to identify at-risk students [12], and to study students' behavior and learning [12].

mor

3 CROWDSORCERER

CrowdSorcerer[1] is an embeddable system designed to first collect student-created assignments and then let students evaluate each others' creations. The students can also write simple input-output tests for their exercises. The gist of the system is to give students the opportunity to reflect on what they have learned, write down assignments, read others' code, and to provide instructors ample assignments for future introductory programming courses.

[1] https://github.com/rage/crowdsorcerer

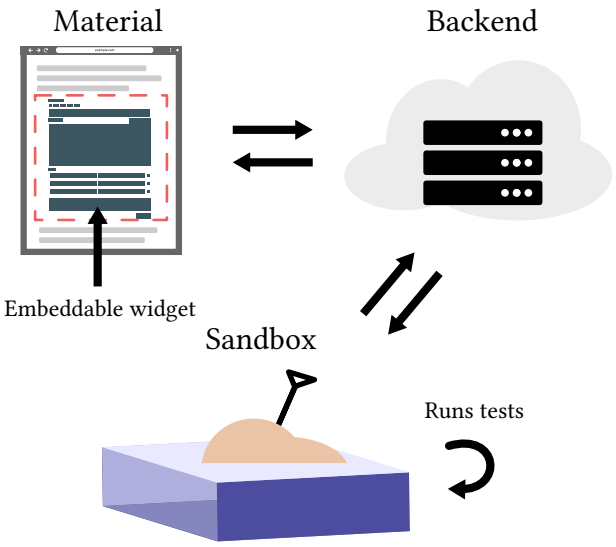

Figure 1: The tool consists of a widget that can be embedded to a webpage, a backend for submissions, and a sandbox server that tests the exercises.

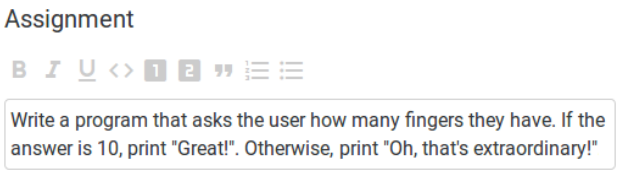

Figure 2: The user specifies instructions for the exercise here. The user is able to for example bold the text and use other enhancements.

3.1 Architecture and design

The students write the instructions for their exercise in a designated field (Fig. 2), and then provide the code for the programming task in the editor with syntax highlighting (Fig. 3). The lines marked as a part of the model solution can be seen in blue, and the boilerplate lines in gray. Lines can be marked by clicking on the line number on the left hand side of the source code view.

The tool consists of a frontend that can be embedded to any online course materials, a backend that stores the data, and a sandbox server that can be used to verify whether the students' code compiles and if the tests work as expected. The frontend is built with React and the backend is built with Ruby on Rails. The frontend uses a REST API provided by the backend to create the exercises and peer reviews. The sandbox has been adapted from Test My Code [23], which is an automated assessment system that can be used to test students' solutions to given programs and to provide feedback to students. The architecture is illustrated in Figure 1.

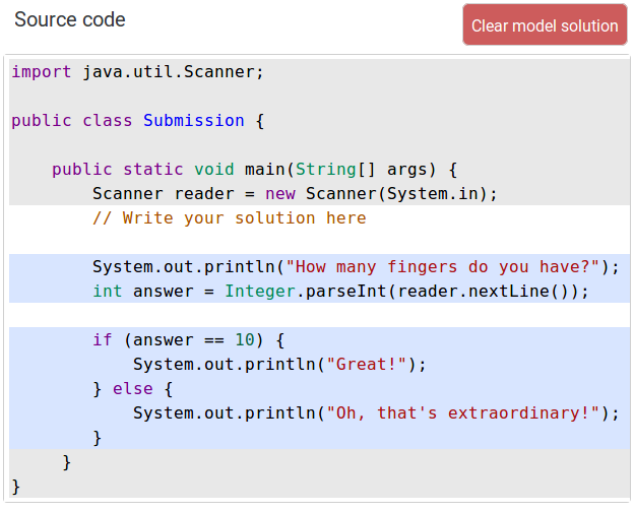

Figure 3: The lines belonging only to the model solution are marked by clicking them, which turns the lines blue. The gray lines indicate boilerplate code.

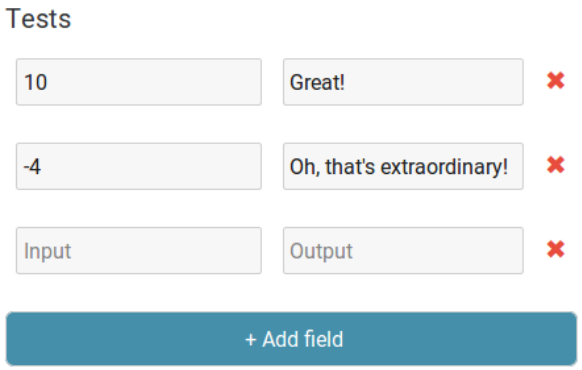

Figure 4: Test cases. The user gives expected inputs and outputs for the completed program. The image has been adjusted slightly to fit the article.

3.2 Creating exercises

Each exercise includes instructions, a model solution, a template, input-output test cases and tags. The student is expected to make the distinction between the model solution and template by "hiding" the most relevant lines of the program. When a line is marked as a part of the model solution (blue lines in Fig. 3), it will not be showed to others once an exercise is downloaded by a user wishing to complete it. The lines which are not hidden form the template. Some tags, defined by course instructors, are recommended, but new ones can also be created freely by ordinary users.

Some boilerplate lines can be defined for the model solution of an exercise. Such boilerplate lines are shown to the user (Fig. 3), but they cannot be modified. The boilerplate typically consists of useful imports as well as a class and a main method definition. Since the tool is intended to be used early on in an introductory programming

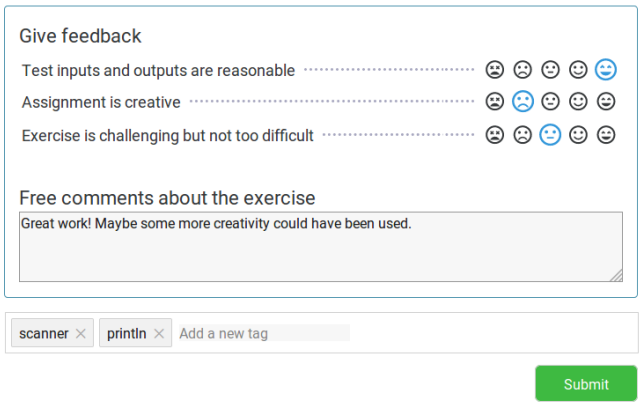

Figure 5: Peer review interface. The user is given the assignment (Fig. 2), source code (Fig. 3) and the tests (Fig. 4) and a list of questions designed by the course instructor, and is supposed to answer them using scale from 1 to 5, illustrated with smiley faces. The user can also add more tags for the exercise.

course, providing the boilerplate code allows students to focus on the key factors that they are working on.

Each exercise that a student generates involves a specific concept, which defines the instructions for the creator of an exercise, the way their program should accept and return information, and peer review questions. For example, an administrator can specify that the exercise should involve for-loops, accept input from the standard input, and output it by printing to the standard output - this way the test runner knows where to look for the output of the program. The exact exercise is then left for the student to define.

After a student has created an exercise, the template and model solution are separated from each other and sent to the sandbox. The sandbox ensures that the template and model solution compile separately and that the model solution prints or returns values as stated in the test cases. The exercise is not marked as completed before the program compiles and all tests pass. The exercises are versioned, so that even if an exercise is completed, it can be resubmitted, and the latest version will be considered in the reviews.

The students are also required to provide at least one test case for their exercise (Fig. 4). Tests are given as input-output combinations that contain the input (possibly as multiple lines) and the output (possibly in multiple lines). By default, the system checks for equality of the output, but it can be also changed to a more relaxed check that verifies that the output lines are (or are not) a part of the output. In the particular example shown in Figure 4, the test cases consist of inputs and their expected outputs. In addition, the students can provide tags or keywords related to their exercise.

The creator of an exercise is notified about the progression with messages such as "Testing model solution" shown above a progress bar. If there are any errors, the system informs the user about it, providing slightly pruned Java error messages and suggests to fix these before resubmission. An exercise cannot be resubmitted without modifications.

3.3 Reviewing exercises

The students can also write peer reviews of the completed exercises through the widget's peer reviewing feature. The administrator-specified general instructions for the assignment are shown as a reminder for the reviewers. The students can download the zipped model solution and exercise template to inspect and run the program locally in a development environment of their choosing, or use the view provided by the tool. The view has two tabs – one for the template, and one for the model solution. The given test cases are also shown and reviewed.

For the peer review process, the tool provides a list of hand-picked review questions that the course instructor can modify and add, such as "The instructions for the exercise are clear", "Template and model solution are clearly divided" and "The exercise is appropriately difficult". In the current version, the answer options are depicted as smiley faces ranging from a sad face to a happy face.

All the submitted exercises and peer reviews can be examined through an administrator interface. The information collected from the assignment submissions includes timestamps, the source code and test cases, exercise description, status (whether the exercise is finished or if there was an error, or in the rare case of server or system failure, some in-between status), possible error messages and information provided by the test server. Administrators and instructors can browse the assignments created by students and see the peer reviews given to them along with a peer-reviewed grade average for the assignment. The peer reviews can also be examined individually.

4 EXPERIENCE REPORT

4.1 Context

The tool was first used on a seven-week CS1 Java programming course during the fall of 2017. The course first covered the basics of programming using an imperative programming style, and then delved into object-oriented programming. The course is primarily targeted at first-year computer science students, but it is also taken by students who study other subjects. Majority of the participants in the course have no previous programming experience. The course material is an online textbook that consists of written theory sections with various topics and exercises which are scattered within the theory sections. The crowdsourcing tool was included in the material similar to the exercises, in-between theory sections.

On the second week of the course, students were asked to design their own exercise according to the instructions given, after which the students were given a prompt to review exercises created by others. Each student was asked to design one exercise, and review three, one of which was their own. The crowdsourcing exercise and peer reviewing process were not obligatory, and no points were awarded, so students could skip these tasks if they chose to do so. The system was not introduced nor discussed in course lectures.

4.2 Sample assignment

Upon the task of creating a programming assignment, the students were given the following description and a brief introductory video that shows how the tool works.

> Create an assignment in which the student is supposed to create a program that reads in an integer from the user, analyzes the input using a set of conditional statements and then outputs a string. In addition to the assignment, write the model solution, tag the rows that need to be hidden from the model solution, and write the tests for the assignment.

4.3 Student responses

From the total of 300 participants in the course, 123 students at least tried to complete the task. From the 123 students, 103 created assignments with model solutions that compiled.

One noticeable issue was that 49 students out of the 123 did not select rows to hide from the model solution that they wrote, leading to a situation where the model solution would be instantaneously given to students who are expected to work on the assignment. As a result, some of the exercise templates and model solutions were exactly the same. It is likely that the concept of a model solution and exercise template were unclear to the students, and future study is required to figure out the best way to mark these lines in this kind of a tool. The current publicly available version of the tool disallows submitting assignments without any model solution lines.

Below we have provided the model solutions from two student generated sample assignments. In these samples, the Java-specific class and method declarations have been omitted. The first one is a rather typical student generated assignment, while the second is an example of an assignment that is too hard considering the desired difficulty level.

```
Scanner reader = new Scanner(System.in);
System.out.println("Planets in our solar system?");
int response = Integer.parseInt(reader.nextLine());

if (response == 8) {
    System.out.println("You got it!");
} else if (response == 9) {
    System.out.println("Forget Pluto, you silly!");
} else {
    System.out.println("Count again!");
}
```

While the assignment above follows the task that was given to students, the assignment below is an example of an assignment where the student chose to introduce loops to the assignment.

```
Scanner reader = new Scanner(System.in);
System.out.println("Type a number ");

int number = Integer.parseInt(reader.nextLine());

if(number == 1) {
    System.out.println("Not a prime number.");
    return;
}

for(int i = 2; i * i <= number; i++) {
    if(number % i == 0) {
        System.out.println("Not a prime number.");
```

```
        return;
    }
}

System.out.println("Optimus prime!");
```

The majority of the student-generated programming assignments followed the task, while a small amount of the students (approx 10-15%) sought to create more complex assignments than expected, which included e.g. the above prime number detector and a rock-paper-scissors game.

4.4 Peer reviews

To assess the suitability of the student constructed programming assignments, each student who had constructed an assignment was then given three assignments to review. From these assignments, two came from others and one from the student themself.

The reviews were given using the peer review functionality of the component, into which we added eight dimensions; (1) reasonableness of test inputs and outputs; (2) assignment matching the solution; (3) assignment clarity; (4) program clarity; (5) appropriateness of the difficulty; (6) sticking to the task; (7) separation between the model solution and the template; and (8) creativity.

A total of 316 peer reviews were given; the majority of the students completed three peer reviews, while a few chose not to do any. If a student had completed the previous assignment, the last of the three exercises was their own, creating the opportunity for self-evaluation. The means and standard deviations for each dimension are given in Table 1. Overall, much of the feedback from students regarding the student generated assignments was positive, most of the dimensions having average scores over 4.4 out of 5.

Appropriateness of the assignment difficulty, separation between the model solution and the template, and the creativity of the assignment were ranked lower. The average for the assignment difficulty dimension was 3.87 out of 5. The notion of "appropriateness" or "suitability" is problematic, as the result can be interpreted in multiple ways; some students may consider the assignment too easy, while others may consider it too hard.

The separation between the model solution and the template (4.01 out of 5) and the creativity of the assignment (3.95 out of 5) were rated low when compared to the rest of the questions. The low score in separation between the model solution and the template is likely explained by the user interface issue in the first version of the tool where 49 students out of the 123 (≈40%) did not separate the model solution and template at all, while the lower value for the creativity is likely explained by the rather simple problem outline: the students were to write an assignment that asks for a number, works with conditionals, and then outputs a string.

Table 1: Peer review scores for student generated assignments. Students were given the opportunity to review others' assignments. In the review, the scale was from 1 (lowest) to 5 (highest) using the faces shown in Figure 5. The averages and standard deviations have been calculated from a total of 316 responses.

Question	Mean	Std.dev.
The test inputs and outputs were reasonable	4.48	0.74
Solution corresponds to the assignment	4.59	0.75
The assignment is clear	4.45	0.74
The code is clear	4.50	0.71
Assignment difficulty is appropriate	3.87	0.97
Assignment asks what was required	4.58	0.77
Separation between the model solution and the template is meaningful	4.01	1.29
The assignment is creative	3.95	0.98

5 DISCUSSION

Crowdsourcing is a great way to allow online courses to scale [24]. While there are existing tools that facilitate crowdsourcing in terms of peer assessment [15] and content creation [4], only a few of the existing tools are explicitly aimed at supporting novice programmers in both creating and evaluating programming assignments. For example, while CodeWrite [5] provides an opportunity to author assignments, the assignments are focused on implementation of methods, and the quantitative reviewing focuses narrowly on the "quality" of the assignment. Similarly, while CloudCoder [16] has authoring and assignments are not restricted to methods, users are expected to be familiar with regular expressions. Both of the above mentioned systems are standalone, and do not currently provide an option for embedding them to course materials.

CrowdSorcerer is be embeddable into online learning materials, reducing the amount of systems that students need to use. It allows the generation of free-form assignments where testing is not restricted to specific method signatures, while still seeking a balance in making the assignment easy to test through a simple input/output -mechanism, which is common in automated assessment [11].

There is still room for improvement when considering the four essential components for open educational resources outlined by Porcello and Hsi [18]. Porcello and Hsi suggest that open educational resources should have common metadata, quality control, community input, and interoperability. In our tool, in the spirit of community input, we have included the students in creating course content. We also hope to get feedback from other educators and researchers working in the computer science education field. Additionally, we control the quality of the assignments the students create by both having the students peer review the assignments they have created as well as having a method for an administrator to review the assignments. Common metadata and interoperability are admirable goals, which a single tool cannot solve by itself. However, an ITiCSE working group from 2014 has sought to address these goals in the computer science education field [2].

As with any system, the ease of use plays a role in how much the system is used. CrowdSorcerer has been designed as an embeddable component that can be used as an integrated part of online course materials. In our experiment, CrowdSorcerer was embedded to the second week of materials in a seven week introductory programming course. While creation of assignments was not mandatory and students did not receive any additional marks for creating assignments, approximately 40% of the students chose to participate

in the activity. At the same time, likely due to a user interface issue, a part of the students created assignments in which the model solution and the assignment template were identical.

Some of the students who created an assignment did not complete any peer reviews, while some of the students who did not create an assignment reviewed three assignments. It might be that providing peer reviews was seen as a less laborious task, or that students have interest in the assignments, even though they did not have the time, creativity or interest to create an assignment themselves. In the experiment, most of the peer reviews were constructive and gave helpful instructions to the creator of the assignment.

As the quality of an assignment is evaluated based on self and peer reviews, the validity of the reviews and given grades should be taken with a grain of salt. In addition, since the reviews are not graded, some students put more effort to their reviews than others. The reliability of peer reviews in the educational context has been studied in for example the medical field [21]. To reduce the possible bias caused by students consistently rating assignments too high or too low, the grades can be normalized in relation to each other, as is done in e.g. the SWoRD system [3].

6 CONCLUSIONS AND FUTURE WORK

In this article, we presented a system called CrowdSorcerer that is used to create and evaluate programming assignments. When compared to previous similar efforts such as CodeWrite [5] and CloudCoder [16], one of the benefits of CrowdSorcerer is that it is embeddable to online learning materials. This means that students do not need to access a separate system or a separate site to use it. Similarly, the tool provides an easy way for the students to include unit tests with the assignment without requiring the students to be proficient programmers – due to this, the tool can also be used as a gentle introduction to software testing, a topic usually encountered later in software engineering related studies.

In our case study, CrowdSorcerer was embedded to the second week course materials of a seven week introductory programming course. While the system was not introduced or discussed in lectures, and students were not given any points or compensated otherwise for using the system, approximately 40% of the students in the course chose to generate at least one programming assignment. During peer evaluations, the meaningfulness and clarity of the assignments and the associated tests were generally rated over four out of five, which suggests that many of the assignments could be used as a part of a programming course.

In our current work, we are both developing CrowdSorcerer and studying its usage. We are currently working on a gradual transition in the test generation process of CrowdSorcerer so that it would support generation of different types of tests. While users currently write simple input-output -tests, future users are able to choose a variety of options for writing the tests ranging from simple input-output -tests to writing actual unit test suites. We are also studying what types of characteristics influence the created assignments: are more proficient students more likely to create (good) assignments, or are there other factors that come into play?

REFERENCES

[1] John Bean and Shevawn Bogdan Eaton. 2001. The psychology underlying successful retention practices. *Journal of College Student Retention: Research, Theory & Practice* 3, 1 (2001), 73–89.

[2] Peter Brusilovsky, Stephen Edwards, Amruth Kumar, Lauri Malmi, Luciana Benotti, Duane Buck, Petri Ihantola, Rikki Prince, Teemu Sirkiä, Sergey Sosnovsky, et al. 2014. Increasing adoption of smart learning content for computer science education. In *Proc. of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*. ACM, 31–57.

[3] Kwangsu Cho, Christian D Schunn, and Roy W Wilson. 2006. Validity and reliability of scaffolded peer assessment of writing from instructor and student perspectives. *Journal of Educational Psychology* 98, 4 (2006), 891.

[4] Paul Denny, John Hamer, Andrew Luxton-Reilly, and Helen Purchase. 2008. PeerWise: students sharing their multiple choice questions. In *Proc. of the fourth international workshop on computing education research*. ACM, 51–58.

[5] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. 2011. CodeWrite: supporting student-driven practice of java. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 471–476.

[6] Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. 2011. Crowdsourcing Systems on the World-Wide Web. *Commun. ACM* 54, 4 (April 2011), 86–96.

[7] Stephen Downes. 2012. Connectivism and connective knowledge. *Essays on meaning and learning networks. National Research Council Canada* (2012).

[8] Enrique Estellés-Arolas and Fernando González-Ladrón-De-Guevara. 2012. Towards an Integrated Crowdsourcing Definition. *J. Inf. Sci.* 38, 2 (April 2012), 189–200.

[9] Edward F Gehringer, Karishma Navalakha, and Reejesh Kadanjoth. 2011. A Student-Written Wiki Textbook Supplement for a Parallel-Architecture Course.

[10] Jeff Howe. 2006. The rise of crowdsourcing. *Wired magazine* 14, 6 (2006), 1–4.

[11] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10)*. ACM, New York, NY, USA, 86–93.

[12] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H. Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, Bronius Skupas, Jaime Spacco, Claudia Szabo, and Daniel Toll. 2015. Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. In *Proc. of the 2015 ITiCSE on Working Group Reports (ITICSE-WGR '15)*. ACM, New York, NY, USA, 41–63.

[13] Chinmay Kulkarni, Koh Pang Wei, Huy Le, Daniel Chia, Kathryn Papadopoulos, Justin Cheng, Daphne Koller, and Scott R. Klemmer. 2013. Peer and Self Assessment in Massive Online Classes. *ACM Trans. Comput.-Hum. Interact.* 20, 6, Article 33 (Dec. 2013), 31 pages.

[14] Leo Leppänen, Juho Leinonen, Petri Ihantola, and Arto Hellas. 2017. Using and collecting fine-grained usage data to improve online learning materials. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*. IEEE Press, 4–12.

[15] Andrew Luxton-Reilly. 2009. A systematic review of tools that support peer assessment. *Computer Science Education* 19, 4 (2009), 209–232.

[16] Andrei Papancea, Jaime Spacco, and David Hovemeyer. 2013. An Open Platform for Managing Short Programming Exercises. In *Proc. of the Ninth Annual International ACM Conference on International Computing Education Research (ICER '13)*. ACM, New York, NY, USA, 47–52.

[17] Nick Parlante. 2007. Nifty reflections. *ACM SIGCSE Bulletin* 39, 2 (2007), 25–26.

[18] Darrell Porcello and Sherry Hsi. 2013. Crowdsourcing and curating online education resources. *Science* 341, 6143 (2013), 240–241.

[19] Kate Sanders, Marzieh Ahmadzadeh, Tony Clear, Stephen H. Edwards, Mikey Goldweber, Chris Johnson, Raymond Lister, Robert McCartney, Elizabeth Patitsas, and Jaime Spacco. 2013. The Canterbury QuestionBank: Building a Repository of Multiple-choice CS1 and CS2 Questions. In *Proc. of the ITiCSE Working Group Reports Conference on Innovation and Technology in Computer Science Education-working Group Reports (ITiCSE -WGR '13)*. ACM, New York, NY, USA, 33–52.

[20] Clifford A Shaffer, Ville Karavirta, Ari Korhonen, and Thomas L Naps. 2011. OpenDSA: beginning a community active-ebook project. In *Proc. of the 11th Koli Calling Int. Conference on Computing Education Research*. ACM, 112–117.

[21] Renée Speyer, Walmari Pilz, Jolien Van Der Kruis, and Jan Wouter Brunings. 2011. Reliability and validity of student peer assessment in medical education: a systematic review. *Medical Teacher* 33, 11 (2011), e572–e585.

[22] Errol Thompson, Jacqueline Whalley, RF Lister, and Beth Simon. 2006. Code classification as a learning and asssessment exercise for novice programmers. *National Advisory Committee on Computing Qualifications* (2006).

[23] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Martin Pärtel. 2013. Scaffolding students' learning using Test My Code. In *Proc. of the 18th ACM conference on Innovation and Tech. in Computer Science Education*. ACM, 117–122.

[24] Daniel S Weld, Eytan Adar, Lydia Chilton, Raphael Hoffmann, Eric Horvitz, Mitchell Koch, James Landay, Christopher H Lin, and Mausam Mausam. 2012. Personalized online education—a crowdsourcing challenge. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*. 1–31.

[25] Wikipedia. 2018. Statistics — Wikipedia, The Free Encyclopedia. (2018). https://en.wikipedia.org/wiki/Special:Statistics [Online; accessed 01-January-2018].